Department I - C Plus Plus

Modern and Lucid C++ for Professional Programmers

Week 15 – Exam Preparation

Thomas Corbat / Felix Morgner

Rapperswil, 12.01.2021

HS2020

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

IFS
INSTITUTE FOR
SOFTWARE

- **Durchführung**
  - Mittwoch 03.02.2021 12:30-14:30 Uhr
  - 4.101 (Aula) / 4.114 / 4.115
- **Rahmenbedingungen**
  - 120 Minuten
  - Fragen sind auf Deutsch
  - Open Book: Papierunterlagen sind erlaubt, ausser alte Prüfungen (Prog3/CPl/CPlA) oder Abschriften davon
  - Keine elektronischen Hilfsmittel
  - Ausweis dabei haben
  - Maskenpflicht

- **Nahe an den Übungsaufgaben / Testaten. Ja, man muss Code lesen und schreiben!**
  - Vererbung mit Ausgabe (Birbs / Monster)
  - Value-Wrapper (Word-Klasse)
  - Iteratoren-Container-Algorithmen
  - Template Aufgabe mit zu adaptierendem Container (IndexableSet)
  - Constness/ADL-Aufgabe
  - Streams and States
  - Value vs. Reference Semantics
- **Multiple-Choice Fragen zu allen Teilen**
- **CUTE Tests muss man lesen und verstehen können**
- **Includes sind wichtig**

- Kommen mehr Verständnisfragen und Quizes wie in den Vorlesungen oder mehr Code schreiben wie in den Testaten? (Wurde in week14 schon mal angesprochen, aber für mich Art der Prüfung immer noch sehr unklar und daher schwer mich Vorzubereiten)
  - Beides: Code schreiben und lesen ist sehr wichtig!
  - Programmieraufgaben sind nahe an den Testataufgaben (vorherige Folie), aber kleiner im Unfang
  - Verständnisfragen meist anhand von Code-Beispielen
  - MC-Fragen möglich
- Werden an der Prüfung includes gefragt? (War bis jetzt in keinem anderen Fach so, darum die Frage)
  - Ja
  - In anderen Sprachen können das die IDEs besser. Wir sind in C++ auf uns alleine gestellt.

- Für die Vergleiches-Ops (= != < > ...): Genügt es die Member-Variante zu kennen oder müssen wird die Andere auch können?
  - Beide Varianten sind wichtig.
- Die Member-Variante macht für diese mehr Sinn oder?
  - Stream-Operatoren können nicht als Member implementiert warden
- Vor/Nachteile?
  - Member-Operatoren haben direkten Zugriff auf die Member
  - Bei Member-Operatoren ist der linke Operand immer das this-Objekt
  - Freie Operatoren können ausserhalb/nachträglich definiert warden

- ("Don't make member variables const as it prevents copy assignment": Warum genau?
  - In C++ ist const sehr strikt. Auch Member-Variablen die const sind können nicht überschrieben werden. Da der Default für Assignment (=) eine Zuweisung für jeden Member macht, ist dies nicht erlaubt.
- Wie tragisch?
  - Verwendung in container eingeschränkt
  - https://godbolt.org/z/WsdPqG
- Wie soll man den Konstanten machen?
  - static, nicht von assignment betroffen
  - global

```
POI createPOI(Coordinate);

auto allPOIs(Coordinate const location) {
  //...
  POI const & migros = createPOI(location);
  //...
  return std::vector{migros};
}
```

Unusual, but correct

`const &` extends the life-time of the temporary POI, until the end of the block. The POI will be copied into the returned `std::vector` object

- What is the deducted type of the `std::vector`? (we expect the type `POI const &`)
  - `std::vector<Coordinate>`
  - Dark magic of template argument type deduction (CPlA)
- Is the referenced object copied into the vector by value or how can the lifetime be extended if the POI initially lived on the Stack?
  - The value is copied. Life-time extension is for the temporary materialized by "`createPOI()`"

- ::isupper als functor aber keine reference auf namespace..
  - isupper ist eine Funktion im globalen Namespace, diese stammt aus dem C-Header <cctype>

```
int countingToLower(std::string& str) {
  auto res = count_if(begin(str), end(str), ::isupper);
  transform(begin(str), end(str), begin(str), ::tolower);
  return res;
}
```

- wieso muss kein namespace angegeben werden für std::function (Argument Dependent Lookup)?
  - std::function kommt in diesem Code nicht vor. Bitte Frage erläutern
- Und was wenn es ein anderer Namespace ist?

```cpp
#include "boost/operators.hpp"
#include <tuple>

class Date : private boost::less_than_comparable<Date> {
  int year, month, day;
public:
  bool operator<(Date const & rhs) const {
    return std::tie(year, month, day) <
           std::tie(rhs.year, rhs.month, rhs.day);
  }
};
```

- Wieso werden boost-mixins private geerbt? Meinem Verständnis nach sollte dies bedeuten, dass ein benutzer meiner Klasse durch das keinen Zugriff auf die geerbten Funktionen erhalten wird. D.h. wieso funktionieren die operatoren beim boost::less_than_comparable?
  - weil boost-mixins nur friends implementieren, welche ausserhalb der Klasse existieren, und deshalb vom public/private keyword nicht beinflusst werden. Stimmt das?
  - Korrekt!

Member Function	Purpose
`begin()` `end()`	Get iterators for algorithms and iteration in general
`erase(iter)`	Removes the element at position the iterator `iter` points to
`insert(iter, value)`	Inserts `value` at the position the iterator `iter` points to
`size()` `empty()`	Check the size of the container

- Common Features of Containers: insert(iter, value)
  Macht dieses Interface sinn im Kontext von nicht-sequence containers? (Zum Beispiel Hashed-Containers)
- **Dies ist nur ein Hint. Bei einem Hashed-Container wird dieser vermutlich ignoriert**
- **Das gemeinsame Interface erlaubt aber die Gleichbehandlung aller Container in generischem Code**

```
template<class InputIt1, class InputIt2, class T>
T inner_product(InputIt1 first1, InputIt1 last1, InputIt2 first2, T init) {
    while (first1 != last1) {
        init = init + *first1 * *first2;
        ++first1;
        ++first2;
    }
    return init;
}
```

- Für parameter by value muss parameter einen Copy/Move constructor bereitstellen?
  - Streng genommen nicht. Der Parameter könnte in-place konstruiert werden
- https://godbolt.org/z/9YPqGf

```
template <typename T>
void foo(T) {
}

struct NonCopyMovable {
    NonCopyMovable(NonCopyMovable const &) = delete;
    NonCopyMovable(NonCopyMovable &&) = delete;
};

int main() {
    foo(NonCopyMovable{});
    NonCopyMovable ncm{};
    foo(ncm);
}
```

- Kann man diesen free_deleter nicht als generisches lambda schreiben? Diese Folie ist sehr schwer zu verstehen, da Function Objects nicht behandelt wurden oder (ist diese Folie dann überhaupt sinnvoll oder eher was für CPl Advanced?). Mit Lambda wäre es aber ev. ein bisschen verständlich (ich habes versucht, bin aber am const_cast gescheitert)

```
auto free_deleter = [](auto p) {
  free(const_cast<std::remove_const<decltype(p)>>(p));
};
```

  - `std::remove_const -> std::remove_const_t`
  - const_cast geht nur auf pointer!

```
auto deleter_lambda = [](auto p) {
   free(const_cast<std::remove_const_t<std::remove_pointer_t<decltype(p)>> * >(p));
};
```

- **Lambda kann nicht instanziiert werden -> muss bei Konstruktion mitgegeben werden**
- https://godbolt.org/z/6dvGoT

```cpp
struct Person : std::enable_shared_from_this<Person> {
  std::weak_ptr<Person> partner;
};

int main() {
  auto person1 = std::make_shared<Person>();
  person1->partner = std::make_shared<Person>();
  std::cout << person1->partner.lock();
}
```

- Anders möglich als mit einer Liste aller Personen, damit für alle ein Shared_ptr existiert, wenn bidirektional ein weak_ptr verwendet werden muss?
  - Ja (?) – Klären ob Frage komplett verstanden

- **When will a template be instantiated?**
  - At compile-time
- **Where do you define a class template and its members?**
  - In the header file? (Usually, completely defined in header). Local definition in source is possible.
- **If you have a member type of another type depending on a template parameter, how do you have to refer to that type?**
  - `typename` Keyword
- **How do you refer to members inherited from a dependent type within a class template?**
  - `this->`
- **Is it neccessary for a (partial) specialization of a class template to provide the same interface (same member functions) as the unspecialized template?**
  - No

- **Special case of List Initialization**
  - If the type is an aggregate, the members and base classes are initialized from the initializers in the list
- Wie verhält sich das im Fall, wenn es zwei Member-Variablen beim Aggregate gibt, beide vom Typ String?
  - String ist kein Aggregate
  - Beispiel mit `int`: https://godbolt.org/z/7rf66M

```
struct Agg {
    int i1;
    int i2;
};

int main() {
    Agg a{1, 2};
}
```

- Wie verhält sich das, wenn das Aggregate von zwei Base-Klassen erbt, welche beide je einen String zur Initialisierung verwenden? Aus meiner Sicht wären dies Spezialfälle, welche wohl mit einem konkreten Initializer implementiert werden müssen.

```
struct IntAg1 {
  int i1;
};
```

```
struct IntAg2 {
  int i2;
};
```

```
struct Compound : IntAg1, IntAg2 {
};
```

```
int main() {
  Compound c{1, 2};
}
```

- https://godbolt.org/z/674TP6

- **Can also provide less initializers than there are bases and members:**
  - If the “uninitialized” members have a member initializer, it is used
  - Otherwise, they are initialized from empty lists

- Selbige Fragen wie oben: Das funktioniert wohl nur, wenn die Zuweisung von Initializer auf Member-Variablen / Basis-Klassen eindeutig ist?
  - Die Reihenfolge ist eindeutig von der Struktur der Aggregates

- Theory Solutions why double brackets for aggregate init? Why is the list encapsulated?
  - Because the member variable in the std::array is a plain C-array, that needs to be initialized with a list.
- How to differentiate direct init and direct list init, without knowledge of the underlying implementation?
  - You need to know the implementation